This application is submitted in the name of inventors Quinn A. Jacobson and Chiao-Mei Chuang, assignors to Sun Microsystems, Inc.

SPECIFICATION

EXPLICIT HIERARCHICAL REGISTER FILE FOR PROCESSORS

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention pertains generally to processor architecture, focussing on the register files used by execution units. More particularly this invention is directed to an improved processor using a hierarchical register file architecture, where the hierarchical register files are visible at the macro-architecture level, facilitating improved performance and backwards compatibility in a processor instruction set.

2. The Prior Art

As reliance on computer systems has increased so have demands on system performance. This has been particularly noticeable in the past decade as both businesses and individual users have demanded far more than the simple character cell output on dumb terminals driven by simple, non-graphical applications typically used in the past. Coupled with more sophisticated applications and internet use, the demands on the system and in particular the main processor are increasing at a very high rate.

As is well known in the art a processor is used in a computer system, where the computer system as a whole is of conventional design using well known components. An example of a typical computer system is the Sun Microsystems Ultra 10 Model 333 Workstation running the Solaris v.7 operating system. Technical details of the example system may be found on Sun Microsystems' website.

A typical processor is shown in block diagram form in FIG. 1. Processor 100 contains a Prefetch And Dispatch Unit 122 which fetches and decodes instructions from main memory (not shown) through Memory Management Unit 110, Memory Interface Unit 118, and System Interconnect 120. In some cases, the

instructions or their operands may be in non-local cache in which case Prefetch And Dispatch Unit 122 uses External Cache Unit 114 to access external cache RAM 116. Instructions that are decoded and waiting for execution may be stored in Instruction Cache And Buffer 124. Prefetch And Dispatch Unit 122 detects which type of instruction it has, and sends integer instructions to Integer Execution Unit 126 and floating point instructions to Floating Point Execution Unit 128. The instructions sent by Prefetch And Dispatch Unit 122 contain register addresses, typically two read locations and one write location, where the read locations are the values to be operated on and the write location is where the result will be stored.

FIG. 1 has one integer and one floating point execution unit. To improve performance parallel execution units were added. One parallel execution unit implementation is shown in FIG. 2. To avoid the confusion and surplus verbiage caused by the inclusion of non-relevant portions of the processor, FIG. 2 and the drawings following it show only the relevant portions of a processor. As will be appreciated by one of ordinary skill in the art, the portion of a processor shown is functionally integrated into the rest of a processor.

A register file, Integer Register File 200, is shown connected to Integer Execution Units 208 and 210 through Bypass Circuit 204. There may be any practicable number of additional integer execution units between Integer

Execution Units 208 and 210. Another register file, Floating Point Register File 202, is shown connected to Floating Point Execution Units 212 and 214 through Bypass Circuit 206. As with the integer execution units, there may be any practicable number of additional floating point execution units between Floating Point Execution Units 212 and 214.

Bypass circuits are needed because it can be the case that one execution unit is attempting to both read a value and write a result to a particular register, or one execution unit may be reading a register in its corresponding register file while another is trying to write to the same register. Depending on the exact timing of the signals as they arrive over the data lines from one or both execution units, this can lead to indeterminate results. Bypass Circuits 204 and 206 detect this condition and arbitrate access. The correct value is sent to the execution unit executing a read, and the correct new value into is written into the register.

The circuitry needed to do this is complex for more than one execution unit, being dependant on the number of register ports attached to one register file. Generally, the complexity of the bypass circuitry rises as the square of the number of register ports a register file has; for n register ports on a register file the complexity of the bypass circuitry rises as n^2.

In addition to the complexity associated with the number of attached execution units and bypass circuitry, a primary bottleneck on the size of register files is the number of ports that must be made available to read and write the registers. The complexity associated with the number of ports is proportional to the square of the total number of ports on a register file. Since there are typically two read operations for every write operation (i.e., most instructions read two values from a register file and write a resulting value), register files typically have two read ports for every write port. If a register file has 8 read ports and 4 write ports, its relative order of complexity would be on the order of $(8 + 4)^2 = 144$ with 12 ports, when compared to other register files with other numbers of ports. Using the same register file and trying to increase its throughput by increasing the number of ports, as an example increasing the number of read ports by 4 and the number of write ports by 2, yields a relative order of complexity of $(12 + 6)^2 = 324$ with 18 ports. As an alternative, adding a duplicate of the original register file yields a relative order of complexity of $(8 + 4)^2 + (8 + 4)^2 = 244$ with 24 ports. Thus, using more register files with fewer ports per register file adds less complexity with more ports (for more throughput) than trying to increase the number of ports on a single register file.

In addition to the complexity just discussed, there are other considerations that limit the size of register files. One problem is physically adding more address and data lines, and the extra length and longer propagation times associated with

the extra length. This is a concern since a register file is usually doubled in size with each increase. The accompanying increase in the number of address and data lines, and the increase in individual lengths and associated propagation delays, run directly counter to the need to increase clock speeds in the processor.

Another problem is addressing the individual registers. To address each of 32 registers in a typical register file requires 5 bits. An example of this addressing may be found in Sun Microsystems UltraSPARC II processor, technical details being available on Sun's website. Each instruction typically has addresses for two values to be read and operated on, and one address to write the resulting value into. Thus, for register files having 32 registers, a total of 15 bits (5 per address) must be allocated per instruction out of a limited number of bits available in each instruction. To add larger register files, for example to make the register files in an UltraSPARC II processor 64 registers long instead of 32 registers, requires that additional bits in each instruction be permanently allocated for addressing. In the case of registers with 64 registers, an additional address bit per address field is needed over register files with 32 registers, for a total of 3 additional bits per instruction. This is a real problem when improvements are being made to an existing architecture. Typically, each word in the existing instruction set is full (all the bits are in use), so no more bits can be allocated to addressing. Even if some instructions have unused bits, it must be the case that the extra address bits be available in all instructions. If they aren't, this causes other problems such as

adding considerable complexity and lack of backward compatibility into microcode.

For the reasons just discussed, adding register file space by increasing the size of the register file is not practical.

In spite of the problems just discussed, the increased parallelism achieved by connecting multiple execution units to one register file has added pressure to increase the number of registers available. Each execution unit may wish to use anywhere from one or more depending on the instructions and operands it is using. This leads to a contention for register space between the execution units, and limits the number that can be connected before there are diminishing returns due to the lack of registers available.

Thus, there are restrictions that necessitate keeping register files at their current size, yet there is a tremendous need for more locally available registers as well.

It is therefore a goal of this invention is to provide a method and system for increasing the throughput of execution units connected to register files by increasing the amount of locally available registers. The goals of increasing the

number of locally available registers in the present invention must be achieved without increasing the size of the register files currently in use.

BRIEF DESCRIPTION OF THE INVENTION

A device and method to increase the throughput of a processor, specifically increasing the throughput of execution units, is disclosed herein. A new architectural feature is added called a backing register file which is directly coupled with the register files, the register files being attached to the execution units in a processor. The backing register file is explicitly visible to users and may be controlled by users. Using the Backing Register File allows users to move values between it and any of the processor's register files, providing a larger register file from which values can be loaded or stored and be ready for immediate use. The Backing Register File may also be used to fetch values from main memory before an execution unit needs them, potentially saving considerable time (preventing stall).

BRIEF DESCRIPTION OF THE DRAWING FIGURES

Figure 1 is a block diagram of a prior art processor.

Figure 2 is a block diagram showing parallelism implemented in a prior art processor.

Figure 3 is a block diagram showing a backing register file according to the present invention.

Figure 4 is a flowchart example of initializing the present invention.

Figure 5 is a flowchart showing use of the backing register file of the present invention.

Figure 6a is a data structure that may be used with the present invention.

Figure 6b is a data structure that also may be used with the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Person of ordinary skill in the art will realize that the following description of the present invention is illustrative only and not in any way limiting. Other embodiments of the invention will readily suggest themselves to such skilled persons having the benefit of this disclosure.

When viewing the figures, it is intended that similar designations used in this disclosure are intended to designate substantially similar matter.

Referring now to FIG. 3, Register Files 308 and 310, and Bypass Circuits 312 and 314 are shown. They perform similar functions as Register Files 200 and 202, and Bypass Circuits 204 and 206. However, due to the extra connections of

Backing Register File 300 the design and implementation will need to be different than the prior art. Integer Execution Units 208 and 210 are shown, potentially having a number of addition integer execution units between them, and Floating Point Execution Units 212 and 214 are also shown and also may have a number of additional floating point execution units between them.

Backing Register File 300 is added to create more local register storage, while not increasing the size of existing Register Files 308 and 310 as compared to Register Files 200 and 202 in FIG. 2. Connection 302 is a full set of address and data lines, allowing Backing Register File 300 the ability to address and access individual registers in each of the Register Files 308 and 310. It will also be the case that the same connectivity will be present between Baking Register File 300 and any register files implemented in a particular processor.

Backing Register File 300 may also be connected to Main Memory 306 through Connection 304. As will be readily understood by those of ordinary skill in the art and with the benefit of the present disclosure, Main Memory 306 is not located on the processor chip and Connection 304 is comprised of a series of connections and interfaces both on and off the chip as more fully described in FIG. 1, with Main Memory 306 being of conventional and well known design. It is expected that cost conscious implementations will not implement a connection between Main Memory 306 and Backing Register File 300, while implementations

where performance has precedence over cost may make use of the extra speed available by having a more direct connection between Backing Register File 300 and Main Memory 306.

Backing Register File 300, being connected to both Register Files 308 and 310, may be used to hold, store, prefetch, and temporarily buffer values in a way that will compliment the number of registers available locally to both the integer execution units and the floating point execution units. This will be particularly useful in holding values that are going to be used again in the instruction stream. By temporarily holding register values that would have been written to main memory considerable time is saved. Another saving occurs when a set of instructions needs to operate on a series of operands but when loading all the operands would preclude other execution units from allocating the space they need for normal execution. It is expected that under normal use, a significant portion (well over half) of the instructions executed by the execution units will not need to make use of Backing Register File 300. Those that do will use Backing Register File 300 as just described, such as for temporary storage instead of using main memory, or to prefetch or preload values into Backing Register File 300 in preparation for execution.

As mentioned above, Backing Register File 300 is especially useful when values would ordinarily have been transferred between execution units and main

memory. Communicating with main memory is long process (many processor clock cycles), which could cause a stall state in one or more execution units as the values are read or written between Register Files 308 or 310 and Main Memory 306. However, with Backing Register File 300 the chances of going into a stall state may be eliminated or at least minimized by using it to temporarily store results, or to hold prefetched values from Main Memory 306 in preparation for the instruction that needs those values. Backing Register File 300 can be used to release execution units and their associated register files as soon as values are written out from Register Files 308 or 310 to Backing Register File 300, and then letting the values in Backing Register File 304 be written to Main Memory 306 using the needed additional clock cycles. This is but one example of how Backing Register File 300 can be used to minimize the time execution units spend being in a stall state, with many more ways of streamlining instruction execution by the use of more registers being readily apparent to those of ordinary skill in the art and having the benefit of the present disclosure.

In a significant departure from the prior art, the present invention crosses the micro-architecture/macro-architecture boundary. Backing Register Files are visible outside the processor and are expected to be explicitly used by programs at all privilege levels. Backing Register File use by programs can take many forms. The two most common usages will be programs complied by smart compilers and, for high performance applications, directly by programmers.

As is well known in the art, sequences of instructions constitute one or more instruction stream or streams, where the instruction streams originate from a program or from more than one program. When used in this disclosure, the concept of a program using the Backing Register includes reference to the instruction stream corresponding to the program from which it originates. In addition, when referring to a program using the present invention "program" includes all programs from any source, including user-originated and system-originated, privileged and non-privileged. When discussing a user-visible instructions contained in a user program, the intent is to include any and all instructions originating from any program, where "user" refers to any program using a processor encompassing the present invention. Thus, "user" is from the processor's view-point where any program uses the processor is a user. This covers the traditional notion of a "user" program which is running on top of (outside of) the operating system, but also includes any other instruction originating from outside the processor – including instructions originating from an operating system or an application-layer program at any level.

Referring now to FIG. 4, a flow diagram shows one way to initialize the use of Backing Register File 300. As a process begins to run, it will send an instruction stream to the processor. The processor will initialize the Backing Register File for use by looking for specific instructions in the instruction stream.

As the instruction flows through the processor, any instructions dealing with the Backing Register File are sent to diamond 400. Diamond 400 checks for the presence of Register Windowing instructions.

Register Windowing is a way of using registers that are not in a Register File. Register Windowing is a legacy of Sun Microsystems in its earlier SPARC processors, further technical information being available from Sun Microsystems on its website. Register Windowing does not have the ability of being able to be randomly accessed over the address space. It uses a base address and makes available a small preset number of registers. Its primary use was to pass parameters for subroutine or function calls. Backing Register File 300 can emulate the behavior of Register Windowing, making Backing Register File 300 backwards compatible with Register Windowing technology and the legacy software that still uses it. Register Windowing emulation capability is a bonus feature of processor architectures that use Backing Register File technology, but is not strictly necessary to practice aspects of the inventive features of the present invention. In an implementation without Register Windowing the steps of 400, 406, and 410 would not be used.

If Register Windowing instructions are found in the instruction stream coming from a process, Backing Register File 300 will be used, together with supporting microcode, to emulate Register Windowing actions. This is shown in

block 406. It will be set in that mode and used that way for the remainder of the time the current process has control of the processor. As soon as the current process no longer has control of the processor, the method will continue back to diamond 400, ready to process further Backing Register instructions.

If no Register Windowing instructions are found, the instruction stream must contain Backing Register File instructions at block 402. This is because there are, basically, only two types of Backing File instructions – one for the Register Windowing capability and one to use the Backing Register File in its native mode. Block 402 is exited to block 408. In block 408 Backing Register File 300 is made fully available to the current process in its native mode. "Native mode" refers to the ability to address each and every register in Backing Register File 300 using its own addresses and at random. When the current process no longer has control of the processor, block 408 is exited and diamond 400 entered, ready to continue processing further Backing Register File instructions.

Referring now to FIG. 5, the process using the processor has sent an instruction making explicit use of Backing Register File 300 and so put the processor in the state shown in step 500 – allowing full access. As the process sends its instruction stream to the processor, each instruction will be checked to see if it is directed to Backing Register File 300 explicitly. If not, step 508 is invoked and the instruction is sent for normal execution. If yes, step 506 is

invoked where the specific action requested in the instruction is either carried out start to finish (e.g., moving a single register value from Backing Register Store 300 to Register File 200) or started (e.g., sending a request for values currently stored in main memory). The instruction determines if it will wait, which puts the execution unit into a stall state if the instruction must wait until its operands arrive. Following step 508, the process begins again at step 502. As will be clear to those of ordinary skill in the art with the benefit of the present disclosure, this illustrative flowchart is not really an endless loop as either the process sending the instruction stream will finish, in which case step 508 is passed but the result of sending the instruction to normal execution terminates the process, or the current process is preempted.

In using Backing Register File 300, a user will issue either some kind of Register Windowing instructions or will request a transfer of register values between register files, main memory, local cache, and Backing Register File 300. This is accomplished using Backing Register File instructions in a program. The data needed to fully accomplish the intended actions will be stored in data structures, and then communicated to the processor using an extended instruction set (Backing Register File instructions recognized in step 504 in FIG. 5).

In the case of the UltraSPARC processor, the standard SPARC instruction set, called SPARC-V9, is documented in *The SPARC Architecture Manual,*

Version 9 and is available from Sun Microsystems. An implementation of the present invention on an UltraSPARC processor would include both the Backing Register File structure disclosed herein and an extended instruction set consisting of instructions that move individual or groups of register values between a backing register file and any register files present, and between a backing register file and main memory. In addition, a set of instructions that emulate Register Windowing would be implemented. The extended instruction set will also have address fields containing enough bits to address the significantly larger address space of a backing register file.

In actual implementation, the extension needed for instruction sets such as SPARC-V9 is very manageable. Only a relatively small number of additional instructions would be needed to make full use of the backing register file. The added instructions would typically have only one source and destination address per instruction, as the new instructions will be "move" instructions rather than "operation" instructions. This means the new instructions will be able to be encoded in the pre-existing instruction length. Thus, to make full use of a backing register file as described and disclosed in the present invention requires an extended instruction set that will be able to make use of the pre-existing instruction length, and will implementable with relatively few new instructions. This constitutes a significant functionality gain with relatively little additional

complexity added in the extended instruction set, constituting another significant advantage of the present invention.

An implementation of the present invention on a non-UltraSPARC processor would include both the device as described above and an extended instruction set consisting of instructions that move individual or groups of register values between a backing register file and all implemented register files, and between a backing register file and main memory, but without instructions that emulate Register Windowing. As stated in the last paragraph, the extended instruction set will have address fields containing enough bits to address the significantly larger address space of whatever size backing register file is implemented.

In the case of a new processor the instructions to direct the working of the Backing Register File would be built into the standard instruction set.

FIG. 6a shows one possible data structure for requesting sets of instructions to be sent to a Backing Register File. There are a set of fields of pre-defined type and length plus a header field, organized as a singly linked list. In this case, the addresses of registers to read or write from the Backing Register File to or from Register File 1 are contained in the first linked field, the addresses to read or write from the Backing Register File to or from Register File 2 are contained in the

second linked field, and so on until the registers to read and write from the Backing Register File to or from Register File n are in linked field n. Another data structure implementation is shown in FIG. 6b, where the linked list with explicit pointers is replaced by a set of fields of specified length, such as two bytes, where every n-th field contains addresses of registers to read or write from the Backing Register File to or from Register File n, and where the entire set of fields is contained in one or two words (e.g., 64 bits which is either two 32-bit words or one 64-bit word).

As will be readily apparent to a person of ordinary skill in the art and having the benefit of the present disclosure, there will be a large number of possible ways of representing the way in which data will be communicated between the Backing Register File and the Register Files, and between the Backing Register File and Main Memory. All such implementations are contemplated by the present invention, and may be used while staying within the spirit of the disclosure.

The present invention relates to processor architecture at both the micro and macro levels, and further relates to an extended instruction set providing explicit (macro level) use of the inventive aspects of the processor architecture. The present invention also encompasses machine readable media on which are stored embodiments (data structures) of the information to be communicated between the

processor and a process using the Backing Register File. It is contemplated that any media suitable for retrieving instructions is within the scope of the present invention. Examples would include magnetic, optical, or semiconductor media.

While embodiments and applications of this invention have been shown and described, it will be apparent to those skilled in the art with the benefit of the present disclosure that many more modifications than mentioned above are possible without departing from the inventive concepts contained herein. The invention, therefore, is not to be restricted except in the spirit of the associated claims.